Operating Systems Project: Topic 6

Investigating and Tuning the Linux Buddy Allocator

Liangsen Wang

224040364@link.cuhk.edu.cn

2026.1.20

Outline

Outline

The Mission: Topic 6 Requirements

Objective: Study and improve the Linux memory allocation subsystem.

Requirement 1: Observation (The "Monitor")

- **Task:** Add instrumentation to the kernel.
- **Goal:** Analyze allocation patterns (e.g., how often are Order-0 vs Order-3 pages requested?).
- **Output:** Logs or visualizations of fragmentation.

Requirement 2: Modification (The "Tuner")

- **Task:** Modify the buddy allocation strategy.
- **Example:** Adjust **Compaction Priority** (aggressiveness of defragmentation) or fallback logic.

The Challenge:
Can you find a contiguous 4MB block in a fragmented heap?

Advanced Options

Option A: Adaptive Buddy System

- **Concept:** Dynamically adjust behavior based on a **Fragmentation Threshold**.
- **Logic:** If fragmentation $> 80\%$, force compaction immediately; otherwise, defer it.

Option B: Slab Allocator Comparison

- **Context:** Buddy manages *Pages* (4KB). Slab manages *Objects* (e.g., 512B).
- **Task:** Compare performance of SLAB vs. SLUB vs. SLOB under different workloads.

Outline

Theory 1: Physical Memory Basics

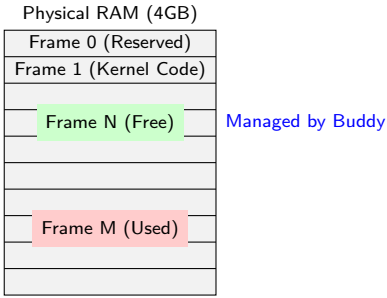
Before we talk about fragmentation, let's understand the raw material: **Physical RAM**.

The Kernel's View:

- RAM is not a byte-array; it is an array of **Page Frames**.
- Standard Page Size: **4KB**.
- Every frame is described by a `struct page` (32-64 bytes overhead).

Why not malloc()?

- `malloc` is a C library function (User Space).
- The Kernel needs a way to allocate memory for *itself* (Page Tables, Process Descriptors).
- **Buddy Allocator** is the Kernel's `malloc`.



Theory 2: Orders and Powers of Two

The Buddy System speaks the language of **"Orders"**. It never allocates 3 pages or 5 pages. It allocates 2^n pages.

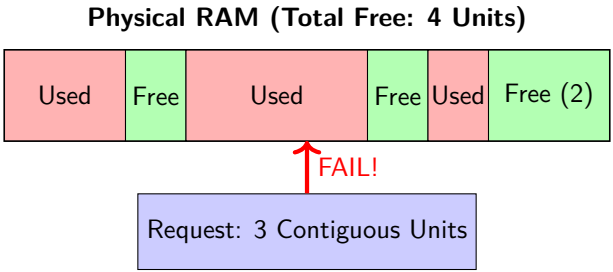
Order	Pages	Size (4KB Base)	Use Case
0	$2^0 = 1$	4 KB	Standard User Page, small buffers
1	$2^1 = 2$	8 KB	Kernel Stack (sometimes)
2	$2^2 = 4$	16 KB	Network Packets (Jumbo Frames)
...	...	...	...
10	$2^{10} = 1024$	4 MB	Hugepages, Video Buffers

The Round-Up Rule

If you need 5KB (1.25 pages), the kernel gives you 8KB (Order 1). This waste is called **Internal Fragmentation**.

Theory 3: The Enemy - External Fragmentation

Definition: Total free memory is sufficient, but no single contiguous block is large enough to satisfy the request.



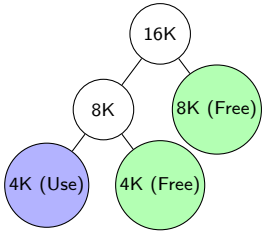
Result: Allocation fails even though we have enough total memory.

Theory 4: The Buddy Algorithm

Philosophy: Split large blocks into halves. Merge halves back when both are free.

Allocation (Request 4KB):

1. Have 16KB block.
2. Split $\rightarrow$ 8KB + 8KB.
3. Split first 8KB $\rightarrow$ 4KB + 4KB.
4. Return first 4KB.

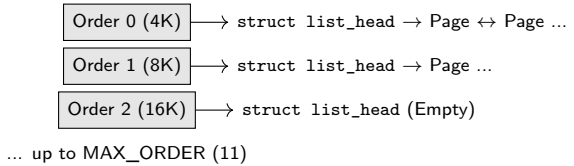


Merging: When the 4K block is freed, the allocator checks its "Buddy" (neighbor). If free, they merge back into 8K.

Outline

Mechanics 1: The Core Structures

Linux organizes memory into **Zones** (DMA, NORMAL). Each Zone has a `free_area` array indexed by **Order** (2^{order} pages).




```c
struct zone {
    struct free_area free_area[MAX_ORDER]; // Usually 11
    ...
};
struct free_area {
    struct list_head free_list[MIGRATE_TYPES];
    unsigned long nr_free;
};

```

Mechanics 2: Memory Compaction (Defragmentation)

When allocation fails due to fragmentation, Linux triggers **Compaction**.

The Project: You can modify *when* this happens (Compaction Priority) or *how hard* the kernel tries before giving up (OOM).

Outline

Step 1: Where to Hack?

The Buddy Allocator lives in `mm/`.

Key File: `mm/page_alloc.c`

This is the Holy Grail.

- `__alloc_pages_nodemask()`: The main entry point for all allocations.
- `get_page_from_freelist()`: The "Fast Path" (try to get a page without waiting).
- `__alloc_pages_slowpath()`: The "Slow Path" (wakes up kswapd, compaction).

Data Structures

- `include/linux/mmzone.h`: Defines `struct zone` and `free_area`.

Step 2: How to Monitor?

You need to see the "Orders" being requested.
Method A: Tracepoints (Recommended) Linux already has `mm_page_alloc` tracepoints. You can hook into them or add your own `printk`.

```
// In __alloc_pages_nodemask (mm/page_alloc.c)
struct page *__alloc_pages_nodemask(gfp_t gfp_mask,
                                    unsigned int order, ...)
{
    // Add your instrumentation here!
    if (order > 0) {
        printk(KERN_INFO "BuddyMonitor: Alloc Order %d\n", order);
        my_fragmentation_stats.count[order]++;
    }
    ...
}
```

Resources & Next Steps

Action Plan:

1. **Read:** `mm/page_alloc.c`. Understand the `order` parameter.
2. **Instrument:** Print logs every time `order > 2` is requested.
3. **Stress:** Write a user program that `mallocs` weird sizes to fragment memory (or use `usemem`).
4. **Visualize:** Read logs and plot "Requested Order Distribution".

Resources:

- *Understanding the Linux Kernel*: Chapter on Memory Management.
- `/proc/buddyinfo`: Shows current free pages per order.